Seamless Data-Rate Change Using Punctured Convolutional Codes for Time-Varying Signal-to-Noise Ratio

Ying Fer a Kar-Ming Cheung

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109-8099

Abstract

In a time-varying signal-to-noise ratio (SNR) environment, symbol rate is changed to maximize data return. However, the symbol-rate changes may cause the receiver symbol loop to lose lock, thus losing real-time data. We propose an alternate way of varying the data rate in a seamless fashion by puncturing the convolutionally encoded symbol stream and transmitting the punctured encoded symbols with a constant symbol rate. We systematically searched for good puncturing patterns for the Galileo (14,1/4) convolutional code and changed the data rates by using the punctured codes to match the Galileo SNR profile of November 9, 1997. We concluded that this scheme reduces the symbol-rate changes from 9 to 2 and provides a larger data return and a higher symbol SNR during most of the day.

1 Introduction

In deep-space communications and other space communications, particularly those requiring multiple antennas for tracking, the combined signal-to-noise ratio (SNR) varies during a day. The degree of variation is determined by weather conditions, antenna elevation angle, antenna pointing accuracy (both the satellite and the ground antennas), changes in satellite latitude, and many other factors. For example, the total signal-power-to-noise-density ratio (P_t/N_o) during a typical 24-hour pass of the Galileo Mission can fluctuate between 15 and 21 dB-Hz. In order to maximize the data return in this time-varying SNR environment, the transmitted symbol rate is varied as a function of the estimated P_t/N_0 at the antenna to maintain a high enough convolutional code symbol SNR. This assures that the bit-error-rate (BER) requirement is met. In the Galileo Mission, there are six different symbol rates, and there can be as many as eight symbol rate changes (from 10 to 640 symbols/second) during a day [1]. One problem associated with the symbol-rate change at low operating symbol SNR

[1] D. Bell and Sung Chiu, "GSAP 3.5 Update/Release", IOM 3392-94-055, May 23, 1994. (Jet Propulsion Laboratory, Pasadena, CA, Internal Document)

is that the symbol synchronization loop may have to reacquire the symbol phase, which may cause real-t,i]l)c data loss. A technique that involve opening the symbol loop at the symbol-rate change has been proposed 2 but this technique require very accurate time predict. It is not clear if the predict information can be obtained within the required accuracy.

In this paper, we are proposing a simple and low-cost alternative solution to the data-rate change problem by changing the data rate at the {rror-correcting coding stage rather than at the transmission stage. The data rate is changed by puncturing the low rate convolutional code while the symbol rate is kept constant. In this way, the basic structures of the encoder and decoder remain unchanged making the scheme simple and less costly. The idea is to minimize the number of symbol rate changes and still maintain a high enough symbol SNR to meet the convolutional code BER requirement. This assures that the carrier loop, the subcarrier loop, and the symbol synchronization loop can remain in lock in the time-varying SNR environment throughout a pass. Symbol rate is changed only if the symbol SNR goes too high (wasting energy) or too low (unable to track the symbols).

In Section 2, we will present the definition and an overview of punctured codes. In Section 3, we will discuss our procedure for selecting good puncturing patterns. In Section 4, we will provide an example of using the punctured convolutional code for the SNR profile of the Galileo Mission 011 November 9, 1997, which is an arbitrarily chosen day. In Section 5, we will give some concluding remarks.

2 Definition and Enumeration of Puncturing Patterns

A regular rate $1/N$ convolutional code requires generating N code symbols per bit. By periodically and systematically refraining from transmission of some of the code symbols, a higher rate code can be constructed from an original lower rate $1/N$ code. This process is called puncturing a rate $1/N$ code to a higher rate. Let the period be 1, bits or NL code symbols. We define a puncturing patter]] P of period NL symbols to be an NL binary tuple where a 1 denotes that the symbol in the corresponding location is to be sent, and a 0 denotes that the symbol is to be deleted. If there are m zeros in 1', the resulting punctured code is a higher rate $L/(NL - m)$ code, where $0 \leq m < (N - 1)L$. For example, let $N = 4$ and $L = 4$, and a puncturing pattern $P = \{01\ 11\ 1110\ 1011\ 1101\}$. We define the rightmost digit to correspond to the first symbol slid the rightmost group of four digits to correspond to the four symbols of the first bit. The puncturing pattern, 1', indicates that the second symbol

"J. Berner, "GLL Data Rate Changes", Notes, June 11, 1993. (let Propulsion Laboratory, Pasadena, CA, Internal Document) in the first bit, the third symbol in the second bit, the first symbol in the third bit, and the fourth symbol in the fourth bit in a period are not transmitted. The resulting punctured code is a rate $4/(4 \times 4 - 4) = 1/3$ code. With the leftmost digit being the most significant bit and the rightmost digit being the least significant bit, the puncturing pattern P can be represented as $dbc7$ in hexadecimal form.

Clearly, there are $\binom{NL}{m}$ different possible patterns for P. Since P is periodic in NL symbols, any cyclic shifts of N symbols in P give the same code performance as P. However, this does not reduce the number of patterns that give a distinct code performance by a factor of N, as some of the $\binom{NL}{m}$ patterns may have a smaller period L_i. That is, L_i divides L, which is denoted by $L_i \mid L$. Let $f(L_i)$ denote the number of puncturing patterns with period L_i exactly (including 1). Notice that $f(L_i) = 0$ if the m zeros cannot be evenly divided among $\frac{L}{L_i}$ partitions (i.e., $\frac{L}{L_i} \nmid m$). Also among the $\binom{NL_i}{\frac{mL_i}{L}}$ patterns with period L_i, some may have smaller periods. Let p be a prime that divides L_i (not including L). If $p \mid \frac{mL_i}{L}$, then there are $\binom{\frac{NL_i}{p}}{\frac{mL_i}{Lp}}$ patterns of P with period $\frac{L_i}{p}$. The total number of distinct puncturing patterns is therefore

$$\sum_{L_i \mid L} \frac{1}{L_i} f(L_i),$$

where $f(L_i)$ can be enumerated as follows:

$$f(L_i) = \binom{NL_i}{\frac{mL_i}{L}} - \sum_{p \mid L_i} \binom{\frac{NL_i}{p}}{\frac{mL_i}{Lp}}$$

Notice that we define the combinatoric function $\binom{m}{n} = 0$ if either m or n is not an integer. In the above example with $N = 4$, $L = 4$, and $m = 4$, an exhaustive search requires checking $\binom{16}{4} = 1820$ puncturing patterns. By taking into account the cyclic property of the puncturing patterns, the search space is now reduced to

:[(3-(31+HW)I+0'4('4]

which is a reduction by almost a factor of 4.

3 Puncturing Pattern Search Procedure

In this section we describe the search procedure that we used to find good puncturing patterns for a rate $1/N$ convolutional code. Using this procedure, we searched for punctured codes for two low-rate convolutional codes, the (14,1/4) code used for Galileo and the (15, 1/6) code used for Cassini. For the (14,1/4) code, we punctured it from rate 1/4 to rate 1/3, then to rate 1/2. For the (15,1/6) code, we punctured it from rate 1/6 to rate 1/5, then to rates

1/4, 1/3, and 1/2. A rate compatibility [1] restriction is added in the puncturing pattern search. That is, a code bit used in the high-rate code is also used in the low-rate code. For example, to search for a rate-1/2 punctured code, we puncture the rate-1/3 code found a step before, not the rate-1/4 code. This was necessary mainly because of limited computing resources. Although we have only searched for puncturing patterns to give rate 1/N codes, those puncturing patterns that give rate k/N codes can be obtained in a similar fashion. Note that a punctured convolutional code of rate k/N is constrained by its low-rate parent code. Therefore, it is not usually as good as the best known general convolutional code of the respective rate. We also present simulation results of the punctured codes and Viterbi decoder. The resulting BERs are used further to compute the BER for the concatenated codes consisting of a convolutional code as the inner code and a Reed-Solomon (RS) code as the outer code, assuming infinite interleaving.

For each described rate where $R > 1/N$, the goal is to find the puncturing pattern P that gives the lowest BER at that rate. Direct simulation of the punctured convolutional code is not viable since there are so many different puncturing patterns. As a first step to select the puncturing patterns, we used the resulting weight profile of the punctured code that includes the maximum free distance, d_{free}, the number of paths of weight d, a_d, and the information bit error weight c_d as criteria. To further simplify our search we only searched for paths of weight d such that $d_{free} \leq d \leq d_{cut}$, where d_{cut} is some pre-determined value that is large enough to infer the code's BER performance and small enough to complete the search in a reasonable time. Note that there are L different starting points for diverging paths, and the worst case is considered in comparing the puncturing patterns.

The aforementioned simplified searching procedure is still computationally intensive. As discussed in the example above, to search for the puncturing pattern that gives the best rate-1/3 punctured code from a (14,1/4) code with period 4, the computation of the weight spectra for the 464 different puncturing patterns takes about 2 days on a SUN SPARC station 10.

The weight spectrum and the free distance are computed for each of the puncturing patterns with period L. From there, three best puncturing patterns are simulated in the convolutional encoder and Viterbi decoder to determine the corresponding BERs for several points of bit-energy-to-noise ratio E_b/N_o. Then the BER of the concatenated code is estimated from the BER of the Viterbi decoder.

Once we have the points of E_b/N_o versus BER, we fit a curve through these points. These curves are used to determine the BER for a given SNR profile of the Galileo Mission. When the BER is less than 10^{-7}, we determine that the code rate can be used in that time period.

The technique for puncturing the codes (14,1/4) and (15,1 /6) used here is rate-colll])atil)lc puncturing [1]. A systematic search is carried out to find the patterns with the maximum free distance and the minimum number of paths of weight d for Viterbi decoding. Three patterns with the largest free distance and smallest number of paths of weight d are then simulated to obtain a bit-error-rate curve. The (14,1/4) Galileo code is used here only to demonstrate the alternative possibility of using punctured codes. In fact, the (14,1/4) code is composed of a (1 1,1/2) convolutional code and the NASA standard (7,1/2) code, and the hardware of the (7,1/2) NASA standard code can not be altered.

3.1 Upper Bound on Free Distance

Before searching for the maximum free distances, we compute the upper bounds of the free distances to serve two purposes, first to verify the results in the maximum free distance calculation, and second to see the effect of the puncturing period on the quality of the punctured codes.

The results show that a lower puncturing period gives a higher upper bound on the free distance, but the lower puncturing period provides fewer choices of code rates.

The upper bounds on the free distances for codes of constraint length K, and rate k/N can be computed using [2],

$$d_f \leq \min_{i \geq I} \lfloor \frac{2^{i-1}}{2^i - 1}(K + i - k)\frac{N}{k} \rfloor \tag{1}$$

where

$$I = \begin{cases} 1, & \text{if } K < 2k - 1 \\ k, & \text{if } K \geq 2k - 1 \end{cases}$$

Figs. 1 and 2 show some of the hounds on the free distances for codes punctured from codes (14,1/4) and (15,1/6) with period from 1 to 8 and 1 to 5 respectively.

3.2 Weight Spectra of Punctured Codes

3.2.1 Code (14,1 /4) as parent code

The parent code in this case has the following polynomials: $2c22$, $3d7d$, $2bcd$, and $1dd3$. First, we search for punctured codes from (1 4,1 /4) to (1 4,1 /3) and find the weight spectra corresponding to all the punctured patterns. The period in this case is 4, which corresponds to 464 different puncturing patterns. We then sort the weight spectra in the ascending order according the number of paths of weight d, a_d. Finally, we simulate the best three patterns to obtain the BER curves, The weight spectra are shown in Table 1. According to the weight spectra, the best pattern is $bbbb$. This implies that the puncturing pattern has period 1, and

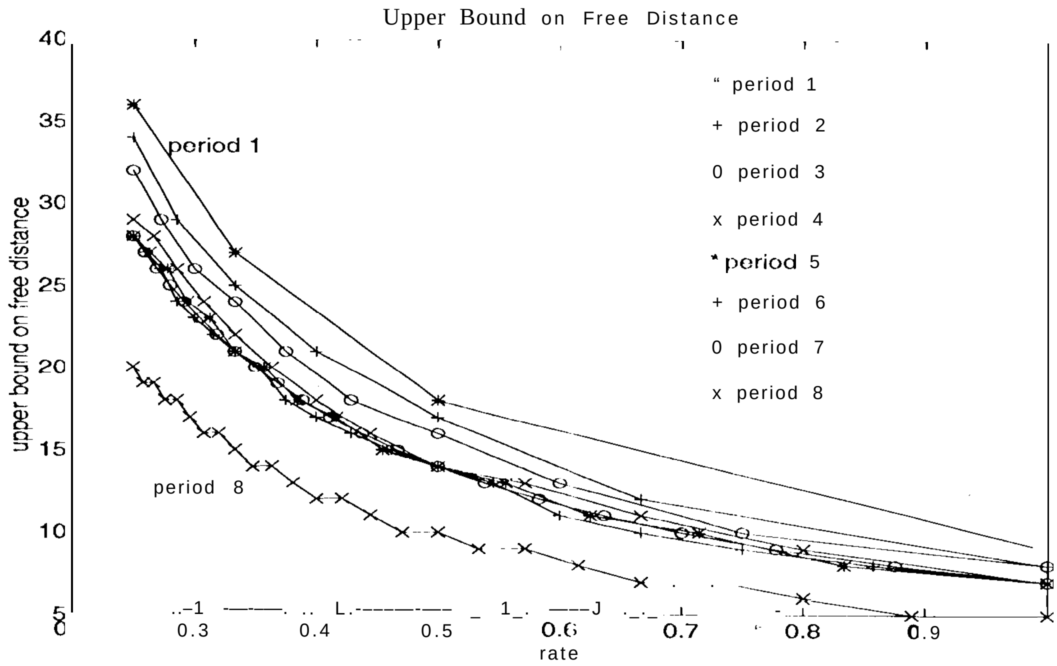


Figure 1: Upper bounds on free distance for punctured codes from (14,1/4) code.

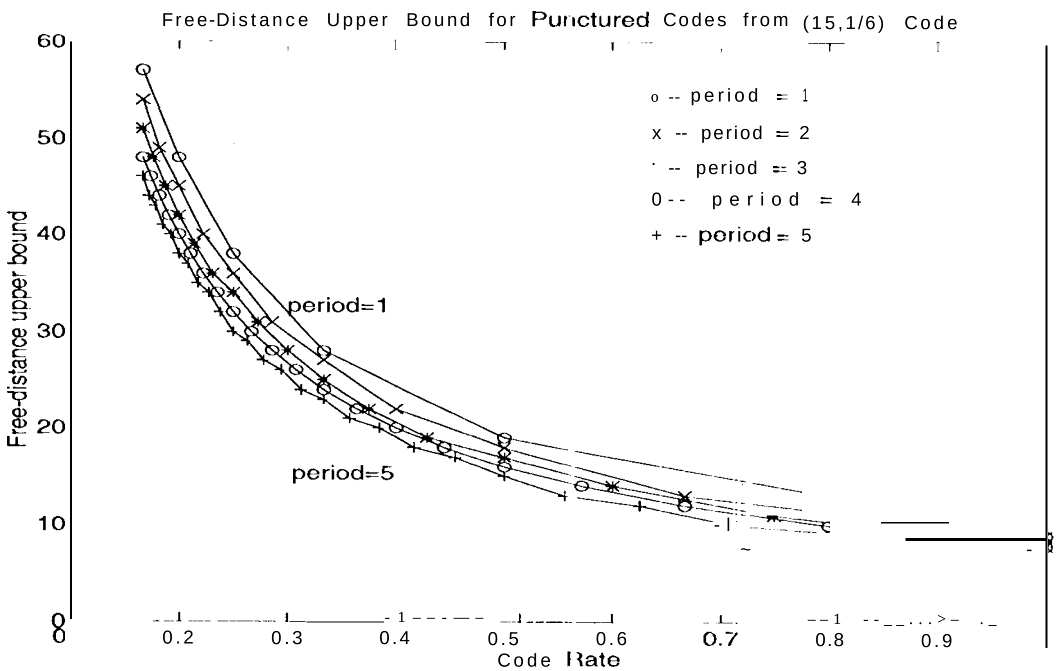


Figure 2: Upper bounds on free distance for punctured codes from (15,1/6) code.

the third symbol is punctured out every time. This corresponds to the (14,1/3) code with polynomials: $2c22$, $3d7d$, and $1dd3$.

To further puncture the code to rate 1/2, we use the best 1/3 code found earlier as the parent code. The following patterns are found to be the best: 3636, 3535, and 3333 in octal numbers. The weight spectra of the three best puncturing patterns are shown in Table 2.

Note that when searching for puncturing patterns with period 4, those patterns with period 1 and 2 are included.

Table 1

Weight Spectra of Punctured Codes (14,1/3) From (14,1/4) Mother Code

pattern	d	23	24	25	26	27	28	29	30	31	32	33	34	35
bbbb	a_d	0	4	6	4	9	14	22	48	93	130	237	389	638
bdbd	a_d	1	1	3	8	13	21	27	54	68	137	225	400	652
bbbd	a_d	1	5	5	5	13	16	38	54	101	146	288	481	800
bbbb	c_d	0	14	18	8	55	72	122	322	641	920	1853	3134	5530
bdbd	c_d	1	3	9	35	60	121	139	320	486	938	1699	3150	5368
bbbd	c_d	5	14	19	21	77	91	240	347	724	1080	2313	4067	7068

Table 2

Weight Spectra of Punctured Codes (14,1/2) From (14,1/3) Mother Code

pattern	d	13	14	15	16	17	18	19	20	21	22	23	24
3636	a_d	0	2	6	10	24	51	142	344	824	1956	4726	11363
3535	a_d	0	2	8	9	35	70	154	371	931	2286	5464	13234
3333	a_d	0	3	0	14	0	73	0	545	0	2884	0	16679
3636	c_d	0	5	20	70	146	354	1144	2914	7780	20229	52967	5525
3535	c_d	0	9	37	53	251	550	1298	3370	9353	25245	64261	357749
3333	c_d	0	9	0	71	0	520	0	4686	0	29299	0	4011

3.2.2 Code (15,1/6) as parent code

We search for puncturing patterns to puncture the code (15,1/6) to (15,1/5) with a period 3. The polynomials for this code are 4599, $4ea5$, $5d47$, $76f3$, $7eb7$, and $695f$. The weight spectra are tabulated in Tables 3-6.

Table 3

Weight Spectra of Punctured Codes (15,1/5) From (15,1/6) Parent Code

pattern	d	46	47	48	49	50	51	52	53	54	55	56	57	58
3bcfb	a_d	0	0	9	0	11	0	18	0	26	0	37	0	79
1f7df	a_d	0	5	5	3	2	5	10	8	12	20	23	26	34
3bcfd	a_d	2	2	6	8	8	8	9	10	12	18	32	32	50
3bcfb	c_d	0	0	25	0	57	0	84	0	130	0	227	0	473
1f7df	c_d	0	15	14	13	10	19	46	38	66	104	130	144	212
3bcfd	c_d	5	10	20	42	39	37	40	63	57	116	197	341	556

Table 4

Weight Spectra of Punctured Codes (15,1/4) From (15,1/5) Parent Code														
pattern	d	32	33	34	35	36	37	38	39	40	41	42	43	44
7bdc	a_d	0	0	0	1	1	3	3	8	7	14	16	17	40
3bfc	a_d	2	0	1	1	3	4	4	6	13	13	23	30	62
2fdf	a_d	2	0	2	1	2	1	8	9	12	22	22	37	33
7bc7	c_d	0	0	0	3	2	15	10	38	38	84	80	95	266
3bfc	c_d	5	0	3	3	16	13	20	28	72	89	170	192	454
2fdf	c_d	6	0	8	4	6	4	46	55	65	139	140	250	221

Table 5

Weight Spectra of Punctured Codes (15,1/3) From (15,1/4) Parent Code														
pattern	d	24	25	26	27	28	29	30	31	32	33	34	35	36
bbb	a_d	0	1	0	3	4	13	16	25	50	68	129	216	378
777	a_d	0	1	0	4	4	11	17	26	47	70	129	202	355
ccc	a_d	0	2	1	5	11	17	30	43	57	94	193	307	530
bbb	c_d	0	3	0	13	16	69	88	161	320	516	1070	1786	3376
777	c_d	0	3	0	18	14	63	98	160	298	516	996	1664	3106
ccc	c_d	0	8	2	27	64	107	206	331	448	744	1632	2791	5012

Table 6

Weight Spectra of Punctured Codes (15,1/2) From (15,1/3) Parent Code														
pattern	d	15	16	17	18	19	20	21	22	23	24	25	26	27
b6d	a_d	0	5	0	43	0	256	0	1279	0	7621	0	42263	0
6db	a_d	0	6	0	29	0	179	0	1044	0	5903	0	32915	0
db6	a_d	0	12	0	35	0	200	0	1234	0	7155	0	40323	0
b6d	c_d	0	19	0	327	0	2445	0	14304	0	30790	0	224	0
6db	c_d	0	34	0	202	0	1558	0	11003	0	5928	0	53499	0
db6	c_d	0	70	0	229	0	1727	0	12563	0	19058	0	11663	0

3.3 BER of Punctured Convolutional Code from Simulation

The weight-spectra search is only the first step in the code puncturing pattern search. To further confirm that the punctured code is non-catastrophic, the punctured codes are simulated with an encoder and the Viterbi decoder for several bit-SNR points. The simulated results are shown in Tables 7 to 12. Generally, the three puncturing patterns give similar BERs when they are non-catastrophic. In Table 12, it can be observed that the puncturing pattern *b6d* results in a catastrophic code.

Table 7

BER of Punctured Convolutional Codes Using Punctured Codes (14,1/3)			
E_b/N_o	BER with punctured patterns		
	bbbb	*bdbd*	*bbbd*
-1.2494	0.3527962	0.3531904	0.3532849
-0.7494	0.2319616	0.2335825	0.2329581
-0.2494	0.1082866	0.1099679	0.1090083
0.2506	0.03423751	0.03451369	0.03474703
0.7506	0.007607836	0.007676672	0.007575335
1.2506	0.001221387	0.001238555	0.001238388

Table 8

BER of Punctured Convolutional Codes Using Punctured Codes (14,1/2)			
E_b/N_o	BER with punctured patterns		
	3636	3535	3333
-1.0103	0.4388964	0.4339047	0.4445088
-0.5103	0.3572518	0.3518962	0.3624895
-0.0103	0.2230175	0.2205774	0.2279394
0.4897	0.09237691	0.09339146	0.09243575
0.9897	0.02297818	0.02142957	0.02259
1.4897	0.003991843	0.004254188	0.00347432
1.9897	0.0005011888	0.0005360237	0.0004340192
2.4897	4.716875e-05	5.383571e-05	3.500155e-05

Table 9

BER of Punctured Convolutional Codes Using Punctured Codes (15,1/5)			
E_b/N_o	BER with punctured patterns		
	3bcfb	*1f7df*	*3bf7d*
-2.2918	0.4472493	0.4474593	0.4468276
-1.7918	0.3793155	0.3802368	0.3798495
-1.2918	0.2619296	0.2641678	0.2651809
-0.7918	0.129610		0.1311644
-0.2918	0.04245451	0.04379791	0.04334172
0.2082	0.009406612	0.009804297	0.009753462
0.7082	0.001616577	0.001670246	0.001648245
1.2082	0.0002125101	0.0002311776	0.0001895509
1.7082	2.366779e-05	2.03343e-05	2.066764e-05

Table 10

BER of Punctured Convolutional Codes Using Punctured Codes (15,1/4)			
E_b/N_o	BER with punctured patterns		
	7bdc	*3dfc*	*2fdf*
-2.2609	0.4605075	0.461228	0.4605616
-1.7609	0.4061571	0.4064082	0.4078231
-1.2609	0.3010911	0.3028663	0.3032402
-0.7609	0.1644293	0.1662707	0.1654938
-0.2609	0.05895112	0.05899296	0.05940715
0.2391	0.01373765	0.01368548	0.01385982
0.7391	0.002347278	0.002310443	0.002297109
1.2391	0.0003761845	0.0003305145	0.0003381827

Table 11

BER of Punctured Convolutional Codes Using Punctured Codes (15, 1/3)			
E_b/N_o	BER with punctured patterns		
	bbb	777	*eee*
-1.5103	0.4154775	0.4170016	0.4242486
-1.0103	0.3150039	0.3176704	0.3242242
-0.5103	0.1757192	0.1778444	0.1784566
-0.0103	0.0640622	0.06504975	0.06300415
0.4897	0.01513205	0.01[illegible]2398	0.01343464
0.9897	0.00262245	0.002286108	0.001921924
1.4897	0.00000000	0.0003196818	0.0002290108

Table 12

BER of Punctured Convolutional Codes Using Punctured Codes (15,1/2)			
E_b/N_o	BER with punctured patterns		
	b6d	*6db*	*db6*
-1.2712	0.5002713	0.4763295	0.4732612
-0.7712	0.5002573	0.4310407	0.425227
-0.2712	0.5005087	0.324346	0.3185326
0.2288	0.4998202	0.1651692	[illegible]
0.7288	0.4995696	0.04732074	0.04973685

3.4 BER of Concatenated Code

Once we obtain the BER from the Viterbi decoder, we can compute the bit-error rate at the output of the Reed-Solomon decoder using

$$P_b = \pi_b \sum_{i=E}^{N-1} \binom{N-1}{i} \pi_s^i (1-\pi_s)^{N-1-i} \qquad (2)$$

where

$$\pi_s = \text{Prob}(\text{J-bit symbol error input to RS decoder})$$

$$\pi_b = \text{Prob}(\text{bit error input to RS decoder})$$

$$N = \text{block length of RS codeword}$$

and

$$E = \text{number of correctable errors}$$

In our special case of Galileo Mission, $J = 8$, $N = 255$, and $E = 16$. The computed BER at the output of the RS decoder is shown in Tables 13 to 15. The parameter π_b is obtained from simulation, and $\pi_s \approx 2\pi_s$.

Table 13

BER Output to RS Decoder Using Punctured Code (14,1/4)		
E_b/N_o	BER input to RS decoder	BER output to RS decoder
-2.0	0.421801	0.421861
-1.5	0.340849	0.340849
-1.0	0.213893	0.213893
-0.5	0.102256	0.102256
0.0	0.0326289	0.01938939
0.5	0.00695925	5.4033257e-9
1.0	0.00128298	2.2583e-20
1.5	0.000178173	1.107975e-49

Table 14

BER Output of RS Decoder Using Punctured Codes (14,1/3)		
E_b/N_o	BER input to RS decoder	BER output of RS decoder
-1.2494	0.3527962	0.3527962
-0.7494	0.2319616	0.2319616
-0.2494	0.1082866	0.1082866
0.2506	0.03423751	0.0222296096098
0.7506	0.0007607836	1.83804e-8
1.2506	0.0001221387	1.005866e-20

Table 15

BER Output of RS Decoder
Using Punctured Code (14,1/2)

E_b/N_o	BER input to RS (decoder)	BER output of RS decoder
-1.0103	*0.4388964*	**0.4388964**
-0.5103	*0.3572518*	**0.3572518**
-0.0103	0.2230175	0.2230175
0.4897	0.09237691	0.09237691
0.9897	0.02297818	0.00293376
1.4897	0.003991843	1.609358e-12
1.9897	0.0005011888	3.683135e-27
2.4897	4.716875e-05	1.609394e-44

4 Example Using Galileo Profile

We use the SNR profile of Galileo mission on November 9, 1997 as an example to explain how the number of symbol-rate changes can be reduced with code rate changes. For a given SNR profile, for example, the one shown in Fig. 3, the objective is to get the maximum data return under the conditions that the bit- error rate is below 1 (1', and that the symbol SNR is maintained above -6 dB for the carrier, subcarr icr, and symbol loops to track. To achieve this goal, the current plan is to change the symbol rate using a fixed code rate, 1/4, and an alternate way is to allow code rate to change as well thus reducing the number of symbol-rate changes. Fig. 4 shows the symbol rates using fixed and variable code rates. In the fixed-code-rate case, there are 9 symbol-rate changes, compared to 2 symbol-rate changes in the variable code rate case. With these symbol rates, each of the two systems will have the symbol SNR above -6 dB as required, where the variable-code-rate case has a slightly higher symbol SNR for most of the day. The code rate changes are shown in Fig. (i. Multiplying the code rates by the symbol rates, we obtain the bit rates as shown in Figs. 7 and 8 for the fixed code-rate and the variable code-rate cases respectively. The bit rates are averaged for the day and the averaged bit rates in both cases are comparable where the variable code-rate case has a slightly higher average bit rate, which implies a slightly larger data return.

5 Conclusions

In this paper we have described a simple and low-cost method to change the data rate to match the time-varying P_t/N_0 environment. This is done by puncturing the convolutional code at the error-correction coding stage rather than by changing the symbol rate at the

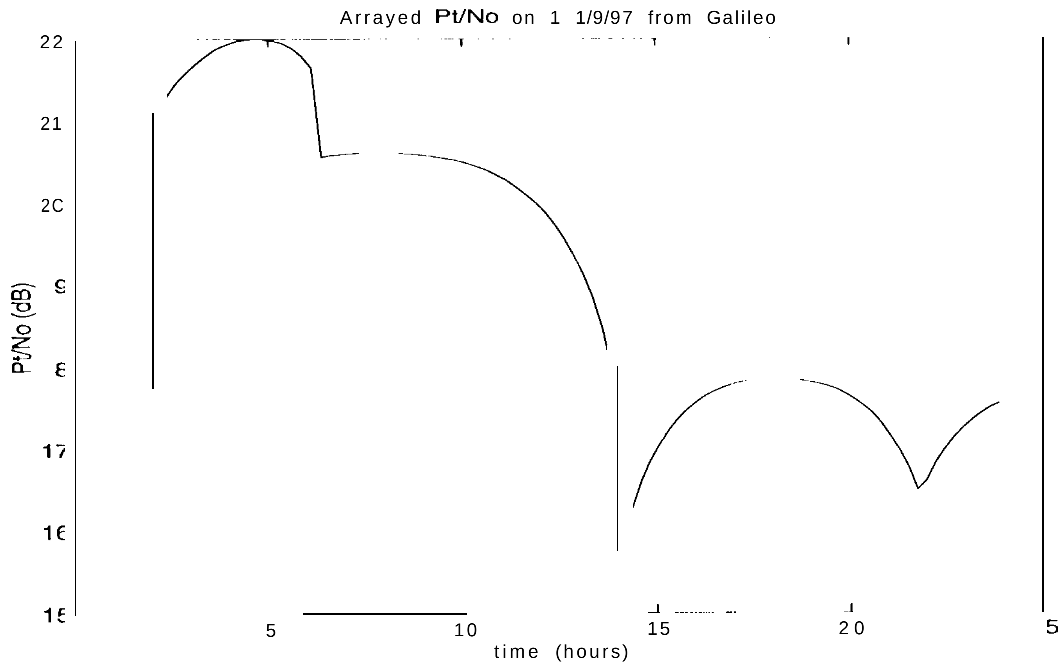


Figure 3: Arrayed P_t/N_o on 11/9/97 from Galileo.

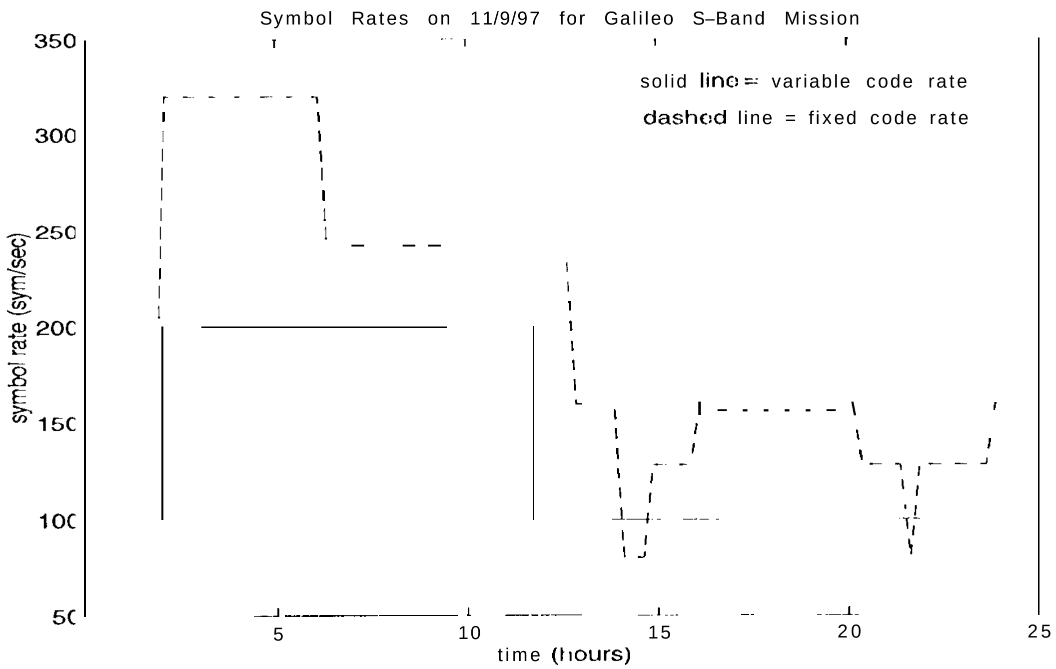
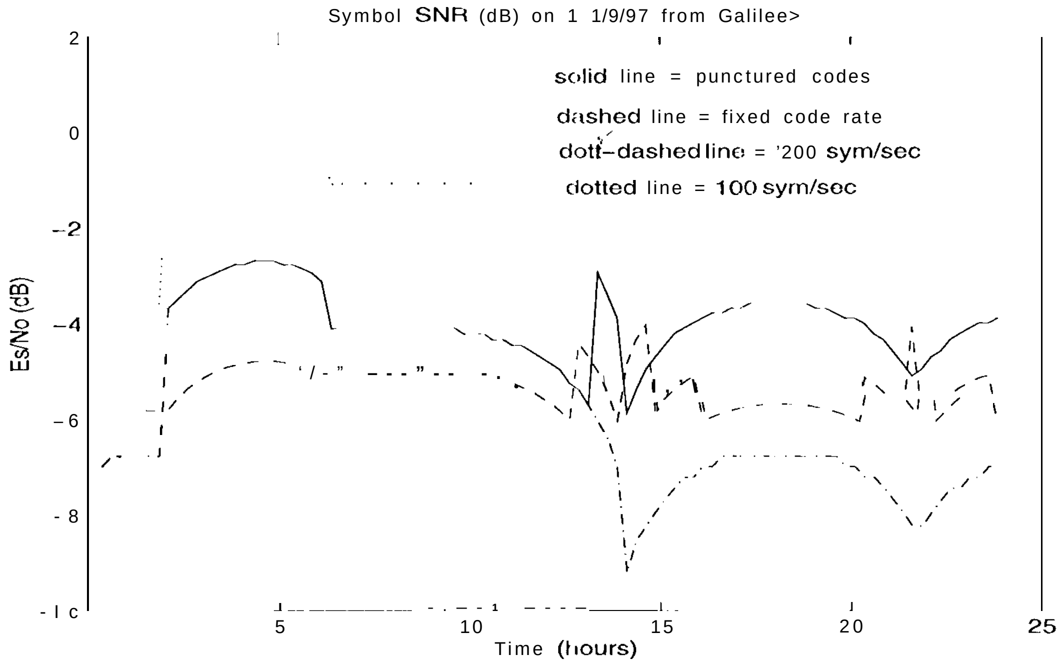


Figure 4: Symbol Rates on 11/9/97 for Galileo using fixed and variable code rates.



dot +
" dotted

Figure 5: Symbol SNR on 11/9/97 for Galileo using fixed and variable code rates

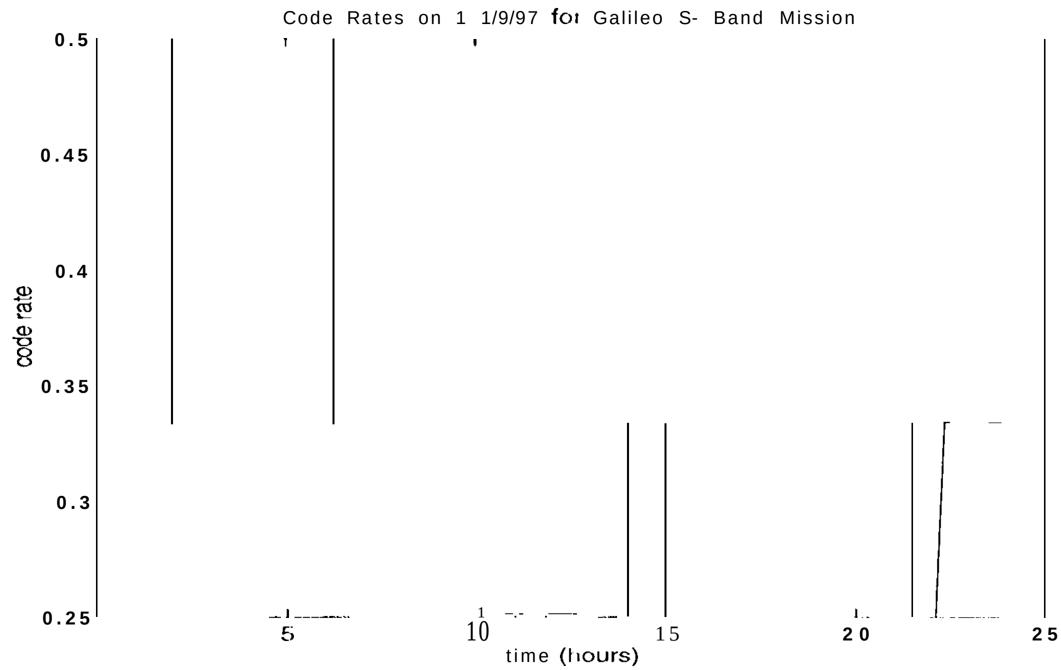


Figure 6: Variable Cede rates on 11 /9/97 for Galileo Mission.

transmission stage. The main advantage of this method is that it allows seamless transition from one data rate to another. We applied this met hod to the Galileo SNR profile ou November 9, 1997 as an example to demonstrate its effectiveness. We showed how this method reduces the number of symbol rate changes from 9 to 2, and gives a slightly larger data return in a day and higher symbol SNR for most of the day. Notice that in this example we arbitrarily pick 1 00 symbols/sec and 200 symbols/sec as the two symbol rates used. We arc working on techniques to pick the pre-selected symbol rates t hat will optimize the data return. We are also working on applying this method to the Cassini communication scenario.

In addition to deep-space communication, there arc other communication environments characterized with a fluctuating P_t/N_0. For example the uncertainty in a geo-synchronous satellite orbit may cause the P_t/N_0 to fluctuate. In mobile communication, fading and interference c-au cause the P_t/N_0 to vary quickly and in great magnitude. This data rate change method can be useful in these applications as well.

Acknowledged

The research and development described in this paper was carried out by Jet Propulsion Laboratory, California Institute of Technology, under a contract with National Aeronautics and Space Administration. Special thanks to Fabrizio Pollara for his multifaceted helps and discussions and for providing software for weight spectra calculation. The authors arc also very grateful to Sam Dolinar for his many suggestions, to David Bell for providing Galileo profile software, and to Todd Chauvin for providing Viterbi decoder simulation tools.

References

1. J. Hagenauer, "Rate-Compatible Punctured Convolutional Codes (RCPC Codes) and Their App lications", *IEEE Trans. on Comm.*, Vol. 36, No. 4, pp. 389-400, April 1988.

2. D. G. Daut, J. W. Modestino, and 1,. D. Wismer, "New Snort Constraint Length Convolutional Code Constructions for Selected Rational Rates", *IEEE Trans. on Inform ation Theory*, Vol. IT-28, No. 5, pp. 794-800, September 1982.